VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Hra demonstrující problém třídění čísel

A Game Presenting the Problem of Sorting Numbers

2018 David Cholewa

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **David Cholewa**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Hra demonstrující problém třídění čísel
A Game Presenting the Problem of Sorting Numbers

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce bude navrhnout a implementovat webovou aplikaci, která bude sloužit pro prezentační účely skupince Teoretické základy informatiky. Aplikace budou zábavnou formou vysvětlovat problém třídění čísel a ukazovat, jak jej lze řešit na počítači. K tomu bude využívat třídící sítě.

Cíle práce lze shrnout v těchto bodech:
1. Seznamte se s třídicími sítěmi.
2. Navrhněte a vytvořte webovou aplikaci, která umožní vizuálně navrhnout třídicí síť a provést simulaci její činnosti. Dále ověří, že skutečně setřídí libovolnou posloupnost čísel.
3. Realizujte možnost zadat třídicí síť pomocí programu v nějakém pseudokódu, případně pomocí nějaké formy vývojového diagramu.
4. Rozeberte, jak upravit vytvořenou webovou aplikaci do podoby hry, kde by hráči mohli soupeřit při vytváření těchto sítí.

Při řešení využijte platformu .NET a programovací jazyk C#.

Seznam doporučené odborné literatury:

[1] Knuth, D. E. (1973). The Art of Computer Programming, volume 3: Sorting and Searching. Addison Wesley.
[2] Griffiths I.: Programming C# 6.0, Create Windows Desktop and Web Applications, O'Reilly Media, June 2015

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Marek Běhálek, Ph.D.**

Datum zadání: 01.09.2017

Datum odevzdání: 30.04.2018

doc. Ing. Jan Platoš, Ph.D.	prof. Ing. Pavel Brandštetter, CSc.
vedoucí katedry	*děkan fakulty*

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 25. dubna 2018 .....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Ostravě 25. dubna 2018 .....

Rád bych na tomto místě poděkoval svému vedoucímu bakalářské práce Ing. Marku Běhálkovi, Ph.D. za odborné rady a čas strávený na konzultacích.

Abstrakt

Cílem této práce je vytvořit webovou hru, která bude sloužit k prezentaci problému třídících algoritmů. Toho je dosaženo tvorbou a vizualizací třídících sítí, jejichž problematika je v této práci rozebrána. Dále se práce zabývá bezkontextovými gramatikami, především LL1 gramatikou, jejíž implementace umožňuje uživateli vložit pseudokód pro vygenerování třídící sítě. Nakonec se práce zabývá samotnou analýzou a implementací aplikace.

Klíčová slova: třídící síť, třídící algoritmus, ASP.NET, LL1 grammar, parser

Abstract

The goal of this thesis is to create a web game, which will be used to present a problem of sorting algorithms. This is achieved by creation and visualization of sorting networks, which are explained in this thesis. The thesis also describes context-free grammars, especially LL1 grammar, which is implemented and allows user to wrtie his own pseudocode to generate a sorting network. In the end this thesis includes analysis and implementation details of the web aplication.

Key Words: sorting network, sorting algorithm, ASP.NET, LL1 grammar, parser

Obsah

Seznam použitých zkratek a symbolů

API	–	Application programming interface
ASP	–	Active Server Pages
CLI	–	Common Language Infrastructure
CLR	–	Common Language Runtime
CSS	–	Cascading Style Sheets
EOF	–	End of File
FPS	–	frames per second
HTML	–	Hyper Text Markup Language
IDE	–	Integrated development environment
JSON	–	JavaScript Object Notation
MVC	–	Model-View-Controller
PHP	–	PHP: Hypertext Preprocessor
XML	–	Extensible Markup Language

Seznam obrázků

Seznam tabulek

1 Úvod

Třídící algoritmy bývají pro začínající programátory představovány jako úvod do algoritmizace. Přestože se dnes dá většinou použít třídící algoritmus z některé z mnohých knihoven, ve výuce programování jsou ukazovány mnohé algoritmy různých složitostí jako ukázka nejenom toho, že jeden problém lze řešit mnohými způsoby, ale hlavně jakým způsobem takové problémy řešit.

Cílem této práce je vytvořit webovou aplikaci, která bude sloužit k představení problému třídících algoritmů. Tato aplikace bude sloužit pro prezentační účely skupiny Teoretické základy informatiky. Aplikace bude tento problém prezentovat zábavnou formou za pomocí třídících sítí.

Třídící sítě jsou výborný způsob, jak představit funkčnost třídícího algoritmu. Lze v nich jednoduše vidět, jak jednotlivé algoritmy pracují a dá se podle nich pochopit jejich funkčnost snadněji než z řádků kódu. Animace třídění čísel taky zaujme pozornost více než obyčejný text. Teorií třídících algoritmů a třídících sítí se zabývá Kapitola 2.

Rozsáhlou částí práce je možnost umožnit uživateli vytvořit třídící síť vlastně vytvořeným pseudokódem. Toho je v této práci dosaženo rekurzivním překladačem, pro který je potřeba vytvořit bezkontextovou gramatiku, která daný jazyk popisuje. Problematika rekurzivních překladačů, bezkontextových gramatik a LL1 gramatiky, která v projektu byla použita, je popsána v Kapitole 3.

Pro tvorbu webové aplikace má být použitá technologie ASP.NET v kombinaci s programovacím jazykem C#. ASP.NET umožňuje zjednodušenou tvorbu klientské a serverové strany a jejich komunikaci. Při tvorbě webové aplikace bylo však použito mnohem více technologií, které jsou spolu s ASP.NET a C# popsány v Kapitole 4.

Kapitoly 5 a 6 se zabývají návrhem a implementací samotné webové aplikace.

2 Třídící algoritmy

Třídící nebo také řadící algoritmus je algoritmus sloužící k seřazení množiny dat do určitého pořadí. Tyto algoritmy pracují na principu přehazování jednotlivých položek pole dat (při splnění podmínky), dokud nejsou data správně seřazena.

Třídění dat se netýká pouze čísel. Data mohou být libovolných typů, dokud se dají mezi sebou nějak porovnávat. Může se např. jednat o třídění slov podle jejich délky nebo podle toho, kolikrát se v daném slově nachází určité písmeno. Princip třídění dat je stejný, ať už se třídí libovolné typy dat. Algoritmy samotné zůstavají v základu stejné, pouze se mění způsob, jakým jsou dané prvky porovnávány.

Teorie třídících algoritmů je velmi rozsáhlá a existuje velké množství ověřených třídících algoritmů. Tyto algoritmy se liší nejenom implementací, ale také rychlostí a jejich ideálním použitím (např. některé algoritmy pracují lépe se skoro setříděným polem než jiné).

Algoritmus	Časová komplexita		
	Nejlepší	Průmerná	Nejhorší
Selection Sort	$\Omega(n^2)$	$\Theta(n^2)$	$O(n^2)$
Bubble Sort	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$
Insertion Sort	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$
Heap Sort	$\Omega(nlog(n))$	$\Theta(nlog(n))$	$O(nlog(n))$
Quick Sort	$\Omega(nlog(n))$	$\Theta(nlog(n))$	$O(n^2)$
Merge Sort	$\Omega(nlog(n))$	$\Theta(nlog(n))$	$O(nlog(n))$

Tabulka 1: Složitosti třídících algoritmů [1]

2.1 Třídící sítě

Třídící sítě jsou speciálním druhem třídících algoritmů, které se značí dvěmi hlavními vlastnostmi:

- Veškeré operace jsou známé dopředu
- Síť pracuje pouze s pevnou velikostí vstupu

Mnohé třídící algoritmy mohou být převedeny na třídící sítě, ale kvůli výše zmíněným podmínkám některé algoritmy převést nelzou.

Třídící sítě se skládají ze dvou hlavních prvků:

- Drát
- Komparátor

Drát slouží k přesunu hodnoty tříděného pole ze začátku algoritmu na konec. Na každém drátu je vždy právě jedna hodnota. Na začátky drátů jsou vložena vstupní data a na konci drátů se nachází výstup třídícího algoritmu.

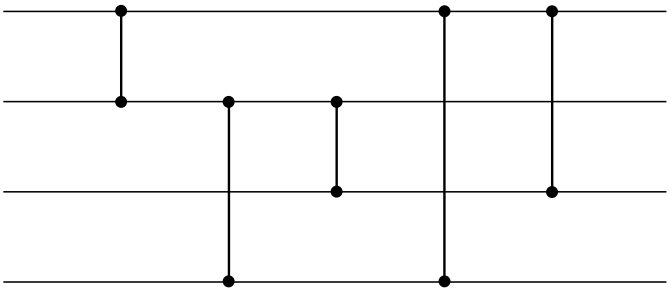
Komparátory slouží k porovnání a přehození dvou hodnot. Každý komparátor propojuje dva dráty v daném kroku algoritmu. Jakmile se hodnoty přesunou ke komparátoru, jejich hodnoty jsou porovnány. Pokud je hodnota na vrchním drátu vyšší než na spodním drátu, hodnoty budou prohozeny.

$$x' = min(x, y)$$
$$y' = max(x, y)$$

kde x a y jsou vstupní hodnoty komparátoru a x' a y' jsou hodnoty výstupní. [2]

2.2 Knuthovy Diagramy

Třídící sítě se dají prezentovat tzv. Knuthovými diagramy. Jedná se o grafické zobrazení, které slouží k přehlednější reprezentaci při studiu třídících sítí. Dráty jsou představovány vodorovnými čárami a komparátory svislými čárami. Vstupní hodnoty začínají vlevo a v průběhu algoritmu se přesouvají doprava, kde se nachází výstup. [2]

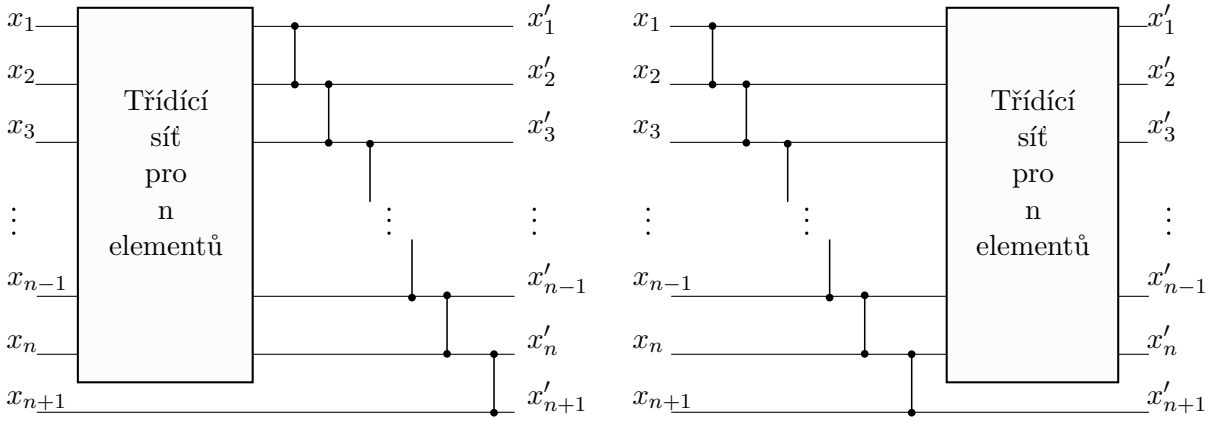


Obrázek 1: Ukázka Knuthova diagramu

2.3 Tvorba třídící sítě

Třídící sítě libovolné velikosti mohou být jednoduše vytvořeny rekurzivně. Nejmenší síť odpovídá jednomu komparátoru, který spojuje dva vstupy. Pro získání sítě dalšího řádu je potřeba správně seřadit další prvek, zatímco předchozí síť se zanechá beze změny. Tento prvek může být seřazen buď před vykonáváním původní sítě (selekce) nebo až po vykonání původní sítě (vložení).

Takto vytvořená síť odpovídá třídícímu algoritmu Insertion Sort, případně Bubble Sort (podle zvoleného typu rozšíření). Nejedná se sice o nejrychlejší nebo nejefektivnější algoritmus, ale jeho velkou výhodou je právě jednoduché vytvoření a rozšíření. [3]



Obrázek 2: Ukázka rozšíření třídící sítě vložením (vlevo) a selekcí (vpravo)

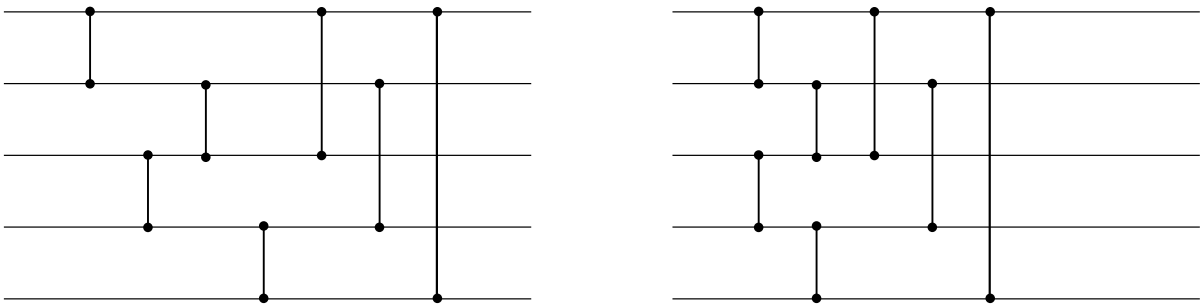
2.4 Vyhledávání největších a nejmenších prvků

Někdy potřebujeme pouze získat největší nebo nejmenší prvek v poli. V takovém případě je zbytečně náročné vytvářet síť, která setřídí celé pole. Jediné, co je potřeba, je vytvoření části sítě, která v Bubble Sortu slouží k seřazení nejposlednějšího prvku.

Pro síť, která vyhledává největší nebo nejmenší prvek je potřeba $n - 1$ komparátorů, kde n odpovídá velikosti vstupu. [4]

2.5 Paralelní třízení

V třídících sítích se mnohdy nachází operace, na jejichž pořadí nijak nezáleží. Jedná se o operace, které se nijak "nekříží". Takové operace lze vykonávat paralelně, takže v jednom kroku může být vykonáno více operací. Tím se celkově zrychlí vykonávání třízení.



Obrázek 3: Ukázka paralelizování třídící sítě

Zde se však nachází jeden problém s Knuthovými diagramy: nechceme, aby se operace křížily. To ve výsledku znamená, že ne všechny operace mohou probíhat paralelně. [4]

2.6 0-1 princip

Pro zkontrolování korektnosti třídící sítě je potřeba vyzkoušet všechny kombinace nesetřízeného vstupu a zjistit, zda budou všechny setřízené. To je však velice náročné, pro velikost vstupu n je potřeba zkontrolovat $n!$ kombinací.

Teorém Z(*0-1 princip*): *Pokud třídící síť dokáže roztřídit libovolnou kombinaci čísel 0 a 1, pak dokáže roztřídit všechny kombinace libovolných čísel.*

Důkaz: Pokud $f(x)$ je monotónní funkce, kdy $f(x) \leq f(y)$ pro každé $x \leq y$, a pokud daná síť transformuje $\langle x_1, \cdots, x_n \rangle$ na $\langle y_1, \cdots, y_n \rangle$, potom lze jednoduše zjistit, že síť transformuje $\langle f(x_1), \cdots, f(x_n) \rangle$ na $\langle f(y_1), \cdots, f(y_n) \rangle$. Pokud $y_i > y_{i+!}$ pro nějaké i, zvažme monotónní funkci f která převede všechna čísla $< y_i$ na 0 a všechna čísla $> y_i$ na 1; to definuje sekvenci $\langle f(x_1), \cdots, f(x_n) \rangle$ nul a jedniček, která není setřízená sítí. Proto pokud všechny 0-1 sekvence jsou setříděné, potom máme $y_i \leq y_{i+1}$ pro $1 \leq i < n$. [5]

To usnadní kontrolu korektnosti sítě, jelikož nyní je třeba zkontrolovat pouze 2^n kombinací čísel.

2.7 Složitost třídících sítí

Složitost třídících sítí se většinou určuje v počtu použitých komparátorů.

Tabulka 2: Složitosti třídících sítí [6]

Algoritmus	Počet komparátorů
Odd-event transposition sort	$O(n^2)$
Bubblesort	$O(n^2)$
Bitonic sort	$O(n \cdot log(n)^2)$
Odd-even mergesort	$O(n \cdot log(n)^2)$
Shellsort	$O(n \cdot log(n)^2)$

Nejnižší rozumná složitost třídící sítí je $O(n \cdot log(n)^2)$. Existují sice algoritmy, které mají složitost $O(n \cdot log(n))$, ale mají své nevýhody. Jeden algoritmus této složitosti je použitelný pouze pro pole mající velikost vstupu větší než 2^{6100}. Druhý z nich zase není korektní pro malé množství vstupů. Obecně použitelný algoritmus mající složitost $O(n \cdot log(n))$ stále nebyl nalezen. [4]

2.8 Použití třídících sítí

Díky své jednoduchosti a podobnosti s elektrickým obvodem mohou být hardwarově implementovány do vestavěných zařízení, které nemají dostatečný výkon pro softwarové řešení.

Třídící sítě jsou dále základem pro kryptografické protokoly. Třídění na třídící síti trvá vždycky stejnou dobu nezávisle na vstupu, díky čemu se např. nedá získat informace o vstupu v závislosti na délce vykonávání algoritmu. [4]

Tato práce využívá třídící sítě pro prezentaci problému třídění množiny čísel. S použitím Knuthových diagramů se jedná o výborný způsob, jak třídění vizualizovat a tvorba sítí je velmi přehledná a jednoduchá.

3 Rekurzivní sestup

Úkolem parseru je analyzovat textový vstup. Jeho hlavním úkolem je zajistit, že zadaný vstupní text splňuje pravidla dané gramatiky, ale během zpracovávání jsou většinou vykonávány i jiné operace (např. výpočet matematického výrazu).

Parsování rekurzivním sestupem je druh parsování, které využívá navzájem rekurzivní pravidla ke zpracování vstupního textu. Tento parser je implementací bezkontextové gramatiky.

3.1 Bezkontextová gramatika

Bezkontextová gramatika je soubor rekurzivních pravidel, které slouží k vygenerování všech slov daného jazyka. Tvorba slov daného jazyka probíhá nahrazováním znaků dalšími pravidly.

$$
\begin{aligned}
S &\rightarrow A|B \\
A &\rightarrow c|\epsilon \\
B &\rightarrow aA|bA
\end{aligned}
$$

Symboly se dělí na terminály a neterminály. Neterminály jsou symboly, které jsou nahrazovány a jsou většinou označovány velkými písmeny. Terminály jsou literální symboly, které se již nenahrazují. Finální slovo gramatiky je sestaveno výhradně z terminálů.

Každé pravidlo určuje, co generuje (značeno znakem „→“) daný neterminál. Na levé straně je neterminál, který je nahrazován, a na pravé je množina terminálů a neterminálů, na kterou je daný neterminál nahrazen. Prázdné slovo je značeno znakem ϵ. Každý neterminál může mít více pravidel. V takovém případě jsou buď zapisovány pod sebe, nebo pro jednduší zápis jsou různá pravidla napsána vedle sebe oddělena znakem '|'. [7]

3.2 LL(1) gramatika

LL(1) gramatika je druh bezkontextové gramatiky, který dle aktuálního vstupního symbolu rozezná následující alternativu. Název LL(1) vychází z:

- vstup je parsován zleva doprava(**L**eft to right)
- vytváří levé derivace (**L**eftmost derivation)
- při volbě dalšího kroku používá pouze **jeden** symbol

Aby gramatika byla LL(1), musí splňovat následující podmínky:

- Gramatika je jednoznačná (v každém kroku se dá jasně rozhodnout o dalším kroku)
- Gramatika není zleva rekurzivní

- Gramatika je deterministická

[8]

3.2.1 Funkce First a Follow

Funkce *First* a *Follow* jsou pomocná pravidla, která slouží k rozhodování, který krok gramatiky je třeba zvolit při analýze.

Funkce *First* označuje množinu terminálů, které se nacházejí na začátku daného symbolu. Funkci *First* jde použít jak na terminály, tak neterminály (oproti funkci *Follow*). Množina funkce $FIRST(X)$ se získá následujícími pravidly:

- Pokud je X terminál, pak $FIRST(X) = X$
- Pokud je X prázdné slovo (ϵ), pak $FIRST(X) = \epsilon$
- Pokud je X neterminál, pak $FIRST(X)$ je množina všech $FIRST(Y) - \{\epsilon\}$, kde Y jsou výsledkem produkce X; pokud je ϵ součástí všech $FIRST(Y)$, pak ϵ také patří do množiny $FIRST(X)$

Funkce *Follow* je definována pouze pro neterminály. Výsledná množina funkce *Follow* je množina terminálů, které se mohou vyskytnout vpravo od daného neterminálu v některé z částečných derivacích. Pokud je neterminál nejpravější symbol částečné derivace, pak do množiny funkce *Follow* patří znak EOF. S množinou funkcí *Follow* se pracuje v případě, že se v množině *First* daného neterminálu nachází prázdný řetězec ϵ. Množina funkce $FOLLOW(X)$ se dá získat následujícími pravidly:

- Pokud je X startovní neterminál, pak EOF patří do množiny $FOLLOW(X)$
- Pro každé pravidlo gramatiky, kde X se nachází na pravé straně:
  - Pro každé pravidlo $Y \rightarrow \alpha X \beta$, kde α a β jsou množiny terminálů a neterminálů, patří $FIRST(\beta) - \{\epsilon\}$ do množiny $FOLLOW(X)$
  - Pokud je ϵ součástí množiny $FIRST(\beta)$, pak $FOLLOW(Y)$ patří do množiny $FOLLOW(X)$
  - Pro každé pravidlo $Y \rightarrow \alpha X$, kde α je množina terminálů a neterminálů, patří $FOLLOW(Y)$ do množiny $FOLLOW(X)$

Pomocí množin *First* a *Follow* lze vytvořit tzv. rozkladová tabulka. Jedná se o tabulku, která jednoznačně říká, jaké následující rozkladové pravidlo se má použít v závislosti na aktuálním neterminálu a následujícím vstupním znaku. V jedné buňce tabulky může být maximálně jedno rozkladové pravidlo. Pokud se v jedné buňce nachází pravidel více, nejedná se o LL(1) gramatiku. Pokud se v buňce nenachází žádné pravidlo, žádné pravidlo pro danou kombinaci neexistuje.

Pokud v takovém případě při parsování k této kombinaci dojde, jedná se o chybu parsování a vstupní textový řetězec není součástí jazyka.

Řádky v rozkladové tabulce značí neterminály, které jsou rozkládány, a sloupce značí následující terminály (spolu se znakem EOF). Tvorba rozkladové tabulky se řídí následujícím algoritmem:

Algorithm 1 Algoritmus tvorby rozkladové tabulky

for Každé pravidlo $X \to \alpha$ **do** ▷ X je neterminál, α je množina terminálů a neterminálů
  for Každý $t \in FIRST(\alpha)$ **do** ▷ t je terminál
    Vlož pravidlo $X \to \alpha$ do $T[X, t]$
  if $\epsilon \in FIRST(\alpha)$ **then**
    for Každý $t \in FOLLOW(X)$ **do**
      Vlož pravidlo $X \to \alpha$ do $T[X, t]$

[9]

Příklad:

Vytvoření rozkladové tabulky pro následující LL(1) gramatiku:

$$
\begin{aligned}
S &\to aA \\
A &\to B|C \\
B &\to DaD \\
C &\to cD|\epsilon \\
D &\to bE \\
E &\to B|C
\end{aligned}
$$

Řešení

Zjistíme množiny *First* a *Follow* všech neterminálů podle předem zmíněných pravidel:

	S	A	B	C	D	E
$FIRST()$	$\{a\}$	$\{b, c, \epsilon\}$	$\{b\}$	$\{c, \epsilon\}$	$\{b\}$	$\{b, c, \epsilon\}$
$FOLLOW()$	$\{EOF\}$	$\{EOF\}$	$\{a, EOF\}$	$\{a, EOF\}$	$\{a, EOF\}$	$\{a, EOF\}$

Tabulka 3: Tabulka *First* a *Follow* množin

V praxi není nutné vytvářet všechny množiny *First* a *Follow*, protože všechny nejsou potřeba pro vytvoření rozkladové tabulky. Zde jsou však vytvořeny všechny množiny pro účel příkladu.

Nyní můžeme vytvořit požadovanou rozkladovou tabulku:

Pohledem na výslednou rozkladovou tabulku 4 si můžeme ověřit, že se skutečně jednalo o LL(1) gramatiku, jelikož se v jedné buňce nenachází více než jedno rozkladové pravidlo.

	a	b	c	EOF
S	$S \to aA$			
A		$A \to B$	$A \to C$	$A \to C$
B		$B \to DaD$		
C	$C \to \epsilon$		$C \to cD$	$C \to \epsilon$
D		$D \to bE$		
E	$E \to C$	$E \to B$	$E \to C$	$E \to C$

Tabulka 4: Výsledná rozkladová tabulka

3.3 Implementace parseru

Při implementaci parseru je nutné vytvořit dvě struktury. Jedna struktura odpovídá množině terminálů a druhá množině neterminálů. Jednotlivé terminály jsou jednoduše implementovány jako třídy. Hlavní funkce objektů těchto tříd je jejich porovnávání s dalšími terminály a případné uložení konkretních hodnot (např. číselná hodnota terminálu představující číslo).

Neterminály jsou implementovány jako funkce. Tyto funkce představují jednotlivá rozkladová pravidla, které patří danému neterminálu. V rámci těchto funkcí jsou postupně jeden po druhém získávány terminály, které představují jednotlivé prvky zpracovávaného textu. V případě, že dané rozkladové pravidlo požaduje konkrétní terminál, je získán terminál vstupního textu a zkontrolován, zda odpovídá očekávanému terminálu. Pokud ano, parsování probíhá dále; pokud ne, parsování skončí neúspěchem. Pokud pravidlo obsahuje neterminál, pak je voláná příslušná funkce, která daný neterminál reprezentuje. Všechna pravidla, která patří danému neterminálu, jsou součástí jedné funkce. Správná funkce se zvolí pomocí *if/else if/else* konstrukce na základě vstupního neterminálu. Celá struktura funkcí neterminálů je vytvořena na základě rozkladové tabulky dané gramatiky.

4 Použité technologie

Součástí zadání je podmínka použití technologií ASP.NET a C#. Role ASP.NET je vytvoření webové aplikace, která je rozdělena na klientskou a serverovou část. Komunikace mezi těmito částmi je zajištěna právě technologií ASP.NET, který v této problematice velice zjednodušuje práci při vývoji. Jako framework pro práci v ASP.NET jsem zvolil Web Forms, především z osobních preferencí. Programovací jazyk C# je použit pro vytvoření chování serveru. Tyto a další technologie jsou popsány v této kapitole.

4.1 ASP.NET

ASP.NET je open-source webový framework vyvinutý společností Microsoft, určený k tvorbě webových aplikací.

Aplikace je rozdělena na dvě hlavní části: klientská část a část serverová. Stránky na uživatelově straně jsou vytvořeny kombinací HTML, CSS a JavaScript kódů. Ve většině případů je konstrukce HTML dokumentu upravena frameworkem tak, aby se ve výsledku chovala tak, jak je zamýšleno. Mezi tyto úpravy patří např. tagy umožňující komunikaci se serverem.

ASP.NET je postavený na technologii .NET, a jako takový může mít kód na straně serveru napsaný v libovolném jazyce, který je podporován technologí .NET (např. C#, Visual Basic .NET, C++/ CLI atd.). Dále je objektově orientovaný a má přístup k .NET knihovnám a vlastnostem jako např. multithreading (aplikace je spuštěná v několika vláknech).

ASP.NET nabízí 3 různé nástroje pro tvorbu webových aplikací: Web Forms, MVC a Web Pages. Každý z těchto frameworků umožňuje jiný způsob práce na webových aplikacích, jsou jinak složité na použití a trochu se liší ve věcech, které umožňují. [10] [11]

4.1.1 ASP.NET Web Forms

ASP.NET Web Forms je eventově zaměřený aplikační model. Umožňuje rychlý vývoj projektu, je poměrně jednoduchý na použití a vhodný pro projekty různých velikostí.

Kromě klasických HTML elementů stránky obsahují speciální Web Forms komponenty (např. Label, TextBox, Button atd.). Tyto komponenty se chovají jako objekty, a jako takové mají svůj vlastní stav a chování. Aby se daly jednoduše tyto komponenty rozlišit od klasických HTML elementů, všechny Web Forms tagy začínají příznakem „asp“(např. <asp:Label> pro textový řetězec). Kromě těchto komponent nabízí Web Forms také možnost vložit do stránky kód, který se vykoná při zpracování. Tento kód je obklopen znaky „<%“a „%>“. Před odesláním stránky uživateli je tento dokument zpracován na straně serveru a je odeslán vygenerovaný HTML dokument obsahující pouze klasické HTML elementy.

Web Forms komponenty mohou reagovat na jednotlivé eventy. Při vyvolání daného eventu je zavolána propojená funkce na straně serveru. Tyto funkce mohou být napsané v libovolném

jazyce .NET (v případě této práce se jedná o C#) a jsou předkompilované, takže zpracování je rychlejší.

Při komunikaci mezi uživatelem a serverem je vždy zasílána celá stránka spolu s údaji popisující aktuální stav komponent, které jsou určené ke zpracování na server, pomocí tzv. „View State“. View State zajišťuje, že server se při žádosti dostane k datům všech komponent, které mohly být na straně klienta editovány, a že klient získá stránku od serveru vyplněnou správnými daty. Velkou výhodou View State je persistence stavu dané stránky v průběhu komunikace mezi klientem a serverem. Komponenta je určena ke zpracování na serveru pomocí atributu "runat=server". [12]

Web Forms již nejsou podporovány v ASP.NET Core (hlavní vývoj ASP.NET momentálně směřuje k technologii MVC).

4.2 C#

C# je objektově orientovaný programovací jazyk vyvinutý společností Microsoft. C# byl oficiálně vydán v roce 2000 a jednalo se o moderní programovací jazyk velmi podobný Javě. V průběhu let C# prošel velkým množstvím změn, od přidání generických typů, lambda výrazů, integrace jazyku LINQ aj. Dnes se jedná o jeden z nejpoužívanějších programovacích jazyků. [13]

V této práci je chování serveru napsáno v programovacím jazyce C#.

4.3 HTML

HTML je značkovací jazyk vzniklý v roce 1990 sloužící k tvorbě webových stránek. HTML popisuje strukturu webové stránky, která řídí, jak bude obsah stránky vysázen, jak se bude chovat a jak bude vypadat.

Obsah stránek je vytvořen strukturou tzv. elementů, které popisují určitý typ obsahu, který se na stránce bude nacházet (např. obrázek, text nebo JavaScriptový skript). Tyto elementy mohou mít atributy konkretizující, jak se daný element má chovat (např. CSS třída nebo cílová adresa odkazu). Většina těchto tagů je párových a vytvářejí stromovou strukturu.

Existují 2 hlavní syntaxe HTML: klasické HTML a XHTML. Rozdíl v XHTML oproti HTML je ten, že se s ním pracuje jako s XML dokumentem. XHTML je striktnější a pokud obsahuje chybu, stránka prohlížečem nebude vůbec vykreslena, zatímco HTML stránka by pouze vynechala chybné elementy. XHTML navíc neobsahuje žádné nepárové elementy. [14] [15]

4.3.1 HTML5

HTML5 je nejnovější verze HTML standardu, který byl oficiálně vydán 28. října 2014. Cílem HTML5 je zjednodušit vývoj webových stránek a nabídnout mnohem více možnosti při jejich tvorbě.

Nové HTML5 elementy se dají zařadit do následujících oblastí:

- Sémantika – rozložení obsahu do různých částí stránky (např. <header>, <footer>, <section>)

- Atributy formulářů – atributy zjednodušující zpracování a validaci vstupu uživatele (např. number, date, calendar)
- Grafické elementy – elementy sloužící pro kresbu (<canvas>, <svg>)
- Multimédia – přehrávání audia a videa bez nutnosti použití pluginů
- Konektivita – komunikace mezi klientem a serverem (např. Web Sockets)
- Storage – ukládání dat na straně uživatele (IndexedDB)
- Výkon a integrace – zlepšení rychlosti složitějších úkolů a lepší práce s hardwarem (např. Web Workers, Drag and Drop, Fullscreen API)
- Přístup k zařízení – použití různých vstupních a výstupních zařízení (např. použití dotykových obrazovek, orientace zařízení)

Dále bylo odstraněno několik elementů. Některé z nich (např. <frame> nebo <dir>) byly odstraněny buď bez náhrady, nebo s možností použí jiný element. Funkce jiných elementů přešla do CSS. [16] [17]

4.4 CSS

CSS je jazyk sloužící pro definici vzhledu elementů webové stránky. Elementy určené k úpravě vzhledu jsou zvoleny pomocí tzv. selektorů. Selektor je vzor, který může obsahovat libovolnou kombinaci tříd, ID, názvů HTML elementů, hodnot atributů atd., přičemž element je zvolen, pokud připadne do daného vzoru.

CSS je pro potřeby dalšího vývoje rozdělen na tzv. moduly. Tyto moduly jsou nezávisle na sobě rozšiřovány a standardizovány. To umožňuje rychlejší přístup nových možností, jelikož není nutné čekat na ostatní části jazyka, pokud je jedna připravena na vydání. [18]

4.4.1 CSS3

CSS3 je nejnovější verze CSS, která přináší velké množství nových možností. Přestože CSS3 jako celek ještě není kompletně hotový, množství již standardizovaných modulů je hojně používáno v moderních webových stránkách. Mezi standardizované moduly patří následující:

- *CSS Color Module* – průhlednost, funkce pro tvorbu barvy (rgb(), rgba(), hsl(), hsla())
- Selektory – substring, nové pseudo-třídy (např. :target, :enabled, :root), sousední elementy ("sibling")
- *CSS Namespaces Module* – podpora XML namespace
- Média – použití CSS stylu na základě uživatelova zařízení

[19]

4.5 JavaScript

JavaScript je objektově orientovaný interpretovaný jazyk určený pro webové stránky. JavaScript je určen pro editaci HTML a CSS, kdy může upravovat hodnoty atributů HTML elementů, měnit jejich styl, nebo může přímo přepisovat strukturu HTML dokumentu (přidávat/odebírat elementy).

JavaScript kód je (spolu s HTML a CSS soubory) odeslán uživateli, a až na jeho zařízení je kód vykonán. Zde se nachází však jeden problém, a to že uživatel může zakázat ve svém prohlížeči vykonávání JavaScriptu. V takovém případě webová aplikace nefunguje tak, jak je požadováno. [20]

5 Návrh

5.1 Specifikace požadavků

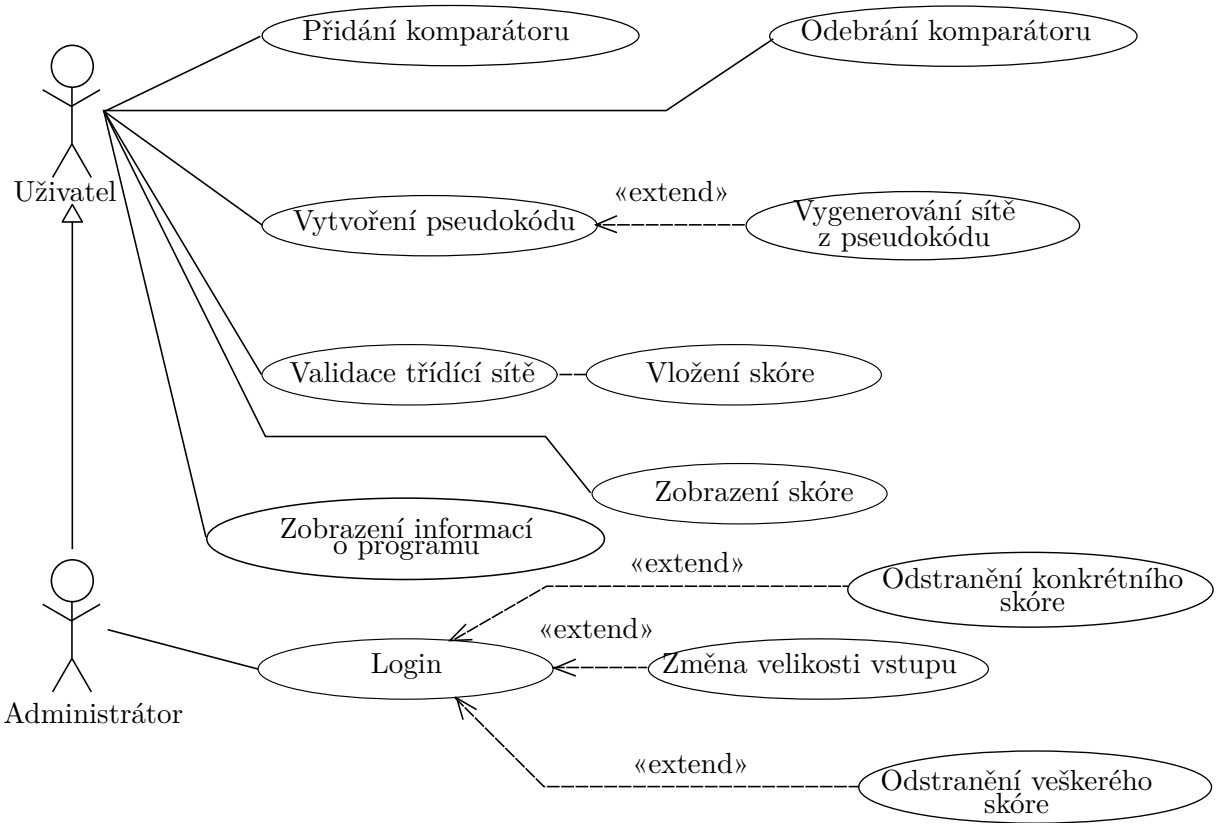
Cílem práce je vytvořit webovou aplikaci, která bude sloužit k prezentaci problému třídících algoritmů. Výsledná aplikace bude prezentována na dnech otevřených dveří pro skupinu Teoretické základy informatiky.

Samotná prezentace problému bude provedena formou třídících sítí. Aplikace bude obsahovat vizualizaci třídících sítí. Tato vizualizace bude sloužit pro grafické zobrazení třídění danými sítěmi a k vytvoření komparátorů třídící sítě. Jelikož bude aplikace sloužit k prezentaci na dni dnech otevřených dveří, aplikace nebude jedním uživatelem používána dlouhodobě. Aplikace proto musí být jednoduše použitelná a pochopitelná, a samotné vytvoření sítě musí být rychlé a intuitivní.

Vytvoření třídící sítě dále bude možné pomocí uživatelem vytvořeného pseudokódu. Bude potřeba vymyslet jazyk, ve kterém bude uživatel vytvářet pseudokód. Pro tento jazyk bude následně vymyšlena bezkontextová gramatika. Tato gramatika bude implementována na straně serveru, kde bude sloužit pro validaci uživatelova pseudokódu a k následnému vygenerování třídící sítě.

Pro zpracování třídící sítě bude vytvořen validátor, který uživatelovu třídící síť zpracuje. Hlavním úkolem validátoru bude zjistit, zda zadaná třídící síť třídí libovolný vstup dané velikosti. Při úspěšné validaci následně vytvoří bodové ohodnocení sítě na základě několika faktorů.

Aplikace bude obsahovat výsledkovou tabulku, ve které se budou nacházet bodová ohodnocení uživatelů.



Obrázek 4: Use Case diagram webové aplikace

5.2 Analýza a návrh

Tato webová aplikace bude rozdělena na část klientskou a část serverovou. Pro vytvoření aplikace bude použit framework ASP.NET s technologií Web Forms. Serverová část bude vytvořena v programovacím jazyce C#, zatímco strana klienta bude vytvořena v kombinaci technologií HTML5, CSS3 a JavaScript. Komunikace mezi serverem a klientem je zajištěna frameworkem ASP.NET.

Klientská strana bude implementovaná jako HTML dokument s *JavaScript* kódy. Hlavní část stránky zabere vizualizace třídící sítě, která kromě animace třízení umožní také ručně vytvářet komparátory. K tomu bude implementována třída, která se bude starat o chování třídících sítí. Jejím úkolem bude vizualizovat třídící sítě na přidělené HTML elementy *canvas*. Spolu s vizualizací také tato třída umožní vytváření komparátorů třídící sítě klikáním na správné pozice příslušného *canvasu*.

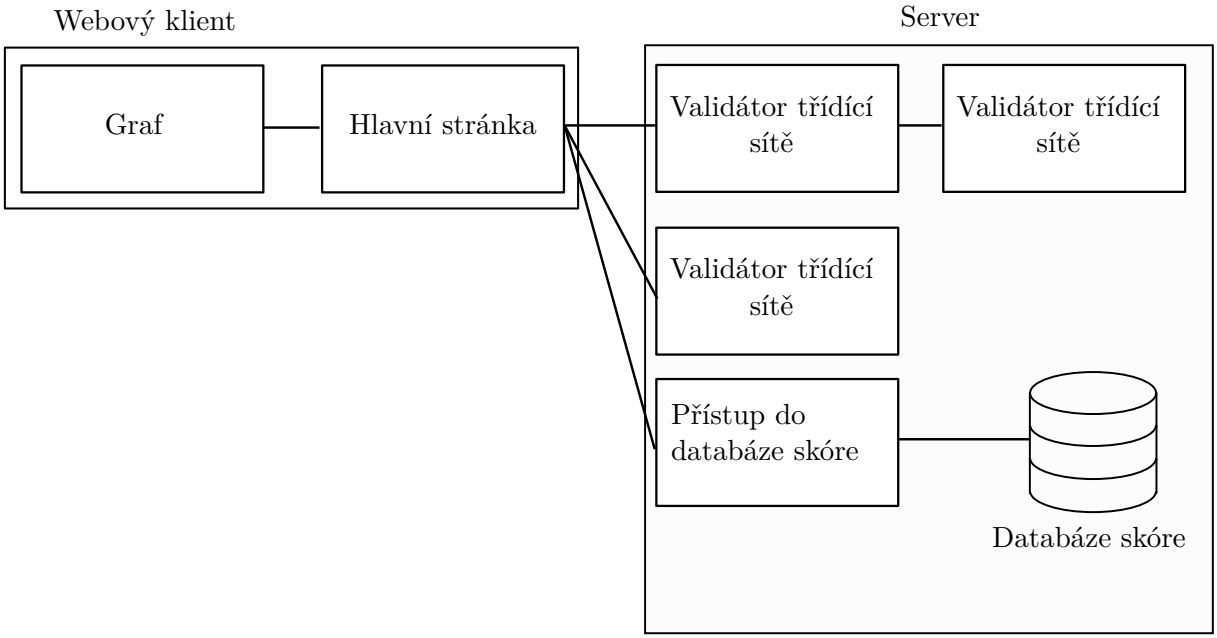
Kromě toho se na stránce budou nacházet další kontrolní prvky, hlavně tlačítka umožňující např. testování třídící sítě nebo žádost o zpracování uživatelem vytvořeného pseudokódu. Tato tlačítka budou ve většině případů propojená s funkcemi na serveru, které při kliknutí budou zavolány.

Na straně serveru se bude nacházet několik velkých částí:

- překladač pseudokódu
- validátor třídící sítě
- databáze skóre

Překladač bude sloužit ke zpracování vstupního pseudokódu, kterým uživatel může vygenerovat třídící síť. Jeho úkolem kromě kontroly, zda je pseudokód validní, je také simulovat spuštění uživatelova pseudokódu a následné vygenerování třídící sítě. Překladač bude implementován jako rekurzivní sestup, přičemž je nutné vytvořit bezkontextovou gramatiku (ideálně LL(1) gramatiku), která popisuje jazyk, ve kterém budou pseudokódy vytvářeny. O simulaci spuštění pseudokódu se postará struktura tříd, která daný pseudokód představuje. Tato struktura bude postupně vytvářena spolu s validací pseudokódu Parserem.

Uživatelova třídící síť je při dokončení zaslána na server pro validaci. O tu se postará validátor třídící sítě. Vstupní třídící síť otestuje, zda je validní, implementací 0-1 principu (viz Podkapitolu 2.6). V průběhu validace navíc vygeneruje statistiky této sítě a finální skóre, které bude závislé na těchto statistikách.



Obrázek 5: Třídní diagram analýzy webové aplikace

Poslední velká serverová část je implementace databáze skóre. Tato databáze bude obsahovat jednotlivá bodová ohodnocení sítí spolu se zadaným jménem uživatele. Tato databáze bude implementována jako jeden XML soubor. Tato databáze bude poměrně jednoduchá, a tak není

nutné používat větší databáze. Práce s touto databází bude zajištěna přes další třídu, která bude umožňovat všechny potřebné operace.

5.2.1 Princip hry

Uživatel má na začátku hry třídící síť obsahující pouze dráty a jeho úkolem je doplnit komparátory tak, aby výsledná třídící síť dokázala roztřídit libovolně setřízený vstup. Komparátory mohou být vytvořeny dvěmi způsoby: uživatel je ručně vytvoří klikáním na graf nebo je nechá automaticky vygenerovat pseudokódem, který napíše.

Vytvořenou síť uživatel zašle na server, kde bude síť zvalidována. Na vytvořené síti poté bude vizuálně prezentováno třídění. Pokud byla síť nekorektní, bude uživateli zobrazené třídění kombinace, která nebude úspěšně setřízena. V případě, že síť je korektní, bude uživateli zobrazena náhodná nesetřízená kombinace. V případě úspěchu je třídící síť bodově ohodnocena a uživateli je nabídnuta možnost vložit skóre do globální tabulky

5.2.2 Grafické rozhraní

Finální aplikace bude sloužit k reprezentaci katedry na dni otevřených dveří. Proto musí být jednoduše a rychle pochopitelná.

Hlavní aplikace se celá nachází na jedné stránce (výjimkou je stránka pro administrátora). Všechny potřebné nástroje pro tvorbu sítě jsou ihned viditelné a přístupné. Největší část obrazovky zabírá graf pro grafické zobrazení sítě jako nejdůležitější prvek aplikace, ale jinak jsou vidět na první pohled všechny prvky pro tvorbu sítě. V případě, že je potřeba zobrazit více informací (např. tabulka skóre, hodnocení sítě nebo informace o aplikaci), je aplikace překrytá popupem (takhle vše zůstane na jedné stránce).

5.2.3 Informace o aplikaci

Aplikace musí obsahovat seznam podrobných informací o webové aplikaci, které si může uživatel přečíst. Bude se jednat o hlavní zdroj informací pro uživatele pro použití programu. Uživatel zde bude moci zjistit množství informací o tom, co je cílem hry nebo kdo je autorem aplikace. Hlavní důvod pro implementaci vestavěných informací o programu je fakt, že uživatel potřebuje získat informace o pseudokódu.

Jelikož je cílem vložit všechny prvky pro uživatele na jednu stránku, tyto informace budou schovány a zobrazeny při stisknutí tlačítka. Tyto informace pak budou překrývat hlavní stránku, s tím že se znovu schová při uzavření informací. To zajistí, že nebude nutné opustit stránku pro přístup k informacím.

5.2.4 Administrace

Tento typ aplikace vyžaduje administrativní režim, ve kterém je možné upravit chování aplikace. Administrátor bude moci schopen v tomto režimu globálně změnit velikost vstupu třídící sítě pro

případné ztížení nebo zjednodušení hry. Dále bude moci administrátor odstraňovat jednotlivé prvky ze skórové tabulky, případně vyčistit tabulku celou. Nakonec bude moci administrátor změnit heslo přístupu.

Administrativní režim se bude nacházet mimo hlavní aplikaci. Pro přístup bude po uživateli vyžadováno přístupové heslo, aby nedošlo k nežádoucímu přístupu. Pro zvýšenou bezpečnost bude administrátor automaticky odhlášen při opuštění stránky.

6 Implementace

Aplikace se dá rozdělit na dva velké celky: klientská strana a strana serveru.

6.1 Hlavní stránka

Struktura stránky je napsána v HTML v kombinaci s ASP *WebControls* prvky. *WebControls* prvky zajišťují jednoduchou práci se serverem. Všechna tlačítka jsou *WebControls* prvky, ale ne všechny vykonávají kód na serveru. Některé z nich (např. tlačítko pro test algoritmu) při kliknutí spustí událost na serveru, který vykoná příslušnou funkci. Při zaslání těchto událostí vždy dojde k obnovení stránky. Jiná tlačítka (např. tlačítko pro pozastavení pohybu čísel v třídící síti) nevolají nic na serveru. To je zajištěno použitím vlastnosti *OnClientClick*. Aby nedošlo k obnovení stránky, kromě zavolání *JavaScript* funkce je dále přidáno „return false; “.

Většinu hlavní stránky zabírají dva HTML elementy typu *canvas*. Tyto elementy slouží k zobrazení třídící sítě a jeho přehledu. Jejich vzhled je inspirován elektronickými obvody.

Logo hry je uloženo jako vektorová grafika ve formátu svg. Takhle je zajištěna větší kvalita při případném zvětšování nebo zmenšování oproti originálu v porovnání s rastrovou grafikou. Dále jsem logo vytvořil ve vektorové grafice kvůli osobním preferencím, takže je možné případně jednoduše později upravit zdroj. Takhle není nutné při změnách neustále obrázek exportovat do rastrové grafiky a nehrozí ztráta originálu.

Tyto dva *canvas* elementy a logo hry se jako jediné nenacházejí v hlavním formuláři, který je zasílán na server. Takhle je ušetřena velikost dat, která je nutná přeposílat ve *ViewState*.

Celá stránka je překrytá několika modaly, které jsou v základu neviditelné. Tyto modaly se zobrazují v případě potřeby (např. zobrazení finálního skóre uživatele). Tyto modaly fungují za pomocí *JavaScript* kódu, pomocí kterého je lze zobrazovat a zpět skrývat. Tento kód jednoduše nalezne modal se správným ID a následně mu nastaví vlastnost *display* na hodnotu *none* nebo *block*.

6.2 Graf

Graf je ten nejdůležitější objekt celé aplikace. Slouží k vizualizaci třídících sítí a jejich ručnímu vytváření. Třídící sítě jsou vizualizované jako Knuthovy diagramy (viz Podkapitolu 2.2). Graf je implementován na straně klienta v jazyce *JavaScript*.

První *canvas* slouží pro vykreslování hlavního grafu. Tento graf má dva hlavní režimy: režim editace a animace. V režimu editace uživatel může vytvářet komparátory klikáním na tento *canvas*. V režimu animace je vizualizováno samotné třízení na síti. Jelikož se jedná o hlavní pohled, ve kterém je možné ručně vytvářet síť, musí být jednoduché na síť klikat i na mobilních zařízeních. Síť je proto zvětšená tak, aby se jednoduše dalo klikat. Pokud je vytvářená síť větší než je *canvas*, část sítě není vykreslována (nachází se mimo canvas). Celá síť pracuje v mřížce (místo konkrétních pixelů) pro jednodušší práci s třídící síti.

Obrázek 6: Screenshot finální aplikace

Pod hlavním grafem se nachází celkový pohled na graf. V něm lze vidět celkový přehled vytvářené třídící sítě. Tento přehled zjednodušuje orientaci v hlavním grafu, protože jdou vždy vidět všechny komparátory, oproti hlavnímu grafu, kde může dojít ke schování komparátorů mimo pohled. Tento graf není určený pro editaci třídící sítě, od toho je hlavní graf. Klikáním na tento graf je však možné přesouvat pohled v hlavním grafu v případě, že je třídící síť příliš velká na hlavní *canvas*.

6.2.1 Zpracování klikání

Hlavní *canvas* zaznamenává eventy kliknutí. Pokud je třídící síť editována, je event zpracován. Jelikož hlavní graf pracuje v mřížce, kliknutí na *canvas* je vždy převedeno na pozici v mřížce (není potřeba pracovat s pixelovou pozicí). Aplikace proto potřebuje vzorce pro převod z pixelové pozice na pozici mřížky a naopak.

$$m_x = Math.round((p_x - o_x - v)/h_x)$$
$$m_y = Math.round((p_y - o_y)/h_y)$$

kde m jsou pozice v mřížce, p jsou pixelové pozice, o je offset mřížky, v je x pozice prvního kroku a h je pixelová vzdálenost mezi dráty, případně kroky. Hodnoty o, v a h jsou předem nastavené konstanty.

Pro připojení pixelové pozice kliknutí na mřížku stačí převést pixelovou pozici na mřížku a následně vypočítat pixelovou pozici pozici dané pozici v mřížce. Při vytvoření nového komparátoru je využita pozice mřížky vypočítaná v průběhu a uložena do nového komparátoru.

Klikáním na druhý *canvas* lze přesouvat pohled hlavního grafu v případě, že třídící síť je tak dlouhá, že se celá na hlavní graf nevleze. Hlavní graf pro vykreslování přebírá hodnotu z objektu přehledu grafu a využívá ji pro úpravu x souřadnic všech vykreslovaných objektů. Tato hodnota se automaticky mění v případě, že probíhá animace třízení a čísla se dostanou za určitou hranicí.

6.2.2 Vykreslování

Pro vykreslování grafu jsou použity dva HTML canvas elementy. Do těchto elementů jsou vykreslovány všechny objekty. Vykreslování probíhá v pravidelných časových intervalech, nastaveno na 30 FPS.

Graf obsahuje množství objektů, které jsou vykreslovány. Některé z těchto objektů (*CircuitCircle*, *DrawableLine* a *GridClickedCircle*) slouží pouze pro vykreslování informací a nebudou v této práci rozebírány. *DrawableComparator* je spojení tříd *Comparator* a *DrawableLine*, jehož úkolem je vizualizovat komparátor podle jeho pozic.

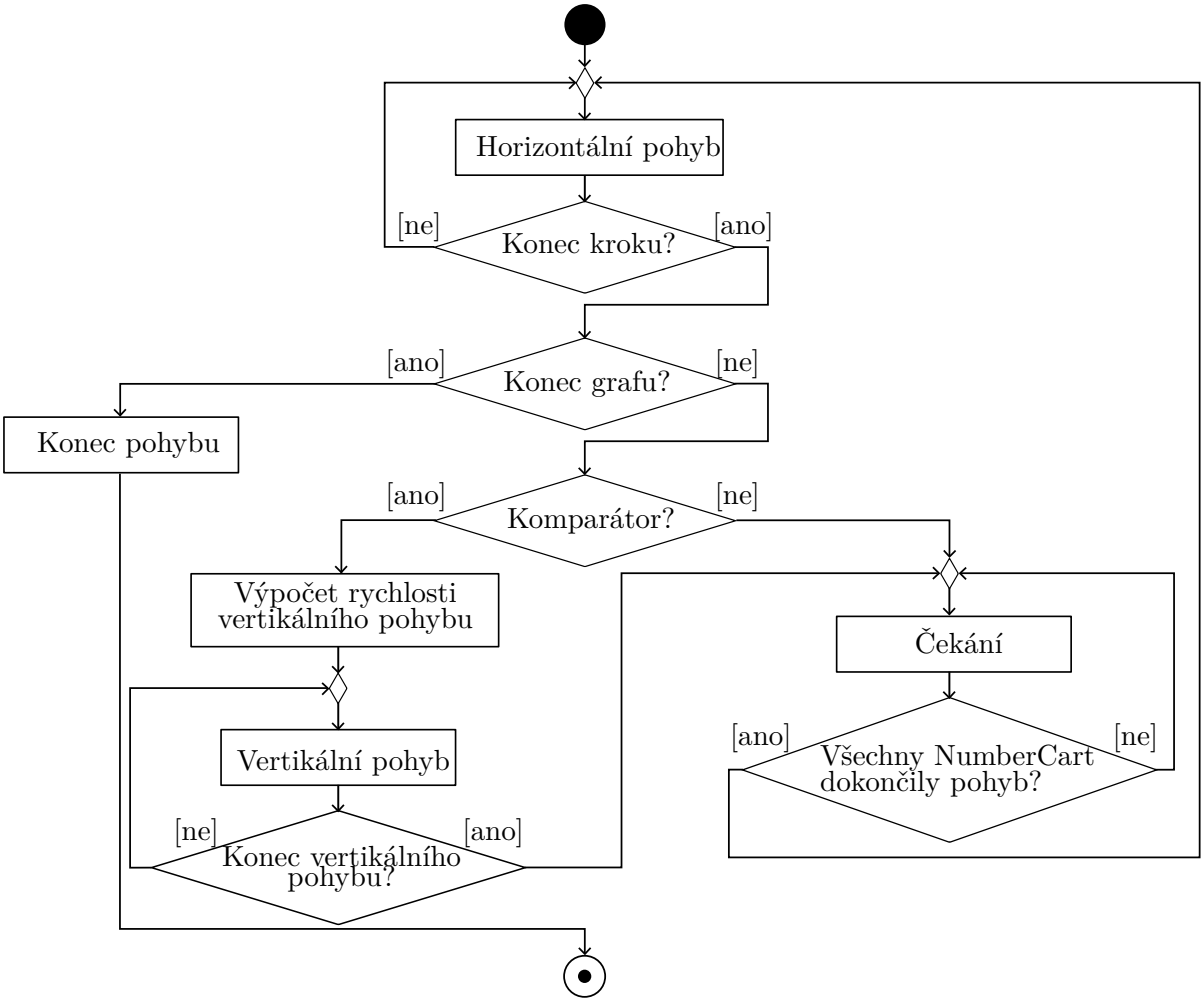
Objekty třídy *NumberCart* slouží k vizualizaci čísel při animaci třídění čísel na třídící síti. Každý *NumberCart* má jednu hodnotu z tříděného pole, které ji je přiřazeno na začátku třídění. *NumberCart* střídá horizontální a vertikální pohyb. Horizontální pohyb probíhá od jednoho komparátoru k druhému. *NumberCart* jsou všechna vždy nad sebou. Jakmile se dostanou k dalšímu kroku třídění, všechny *NumberCart*, které se nacházejí na místě, které je propojené s komparátorem, se začnou vertikálně pohybovat k druhému bodu komparátoru. Ostatní komparátory čekají, dokud vertikální pohyb ostatních komparátorů neskončí. Pak se pokračuje horizontálním pohybem. Celý pohyb skončí jakmile všechny *NumberCarts* objekty dojedou na konec třídící sítě, který se nachází jeden krok za posledním komparátorem.

Všechny objekty vykreslovány na hlavním grafu jsou vykreslovány se stínem. Aby celá síť vypadala jako jeden objekt, jsou první vykreslovány stíny zvlášť (stejný objekt vybarvený černou barvou, která je o několik pixelů posunutý) a následně samotné objekty.

6.3 Parser

Uživatel může vygenerovat třídící síť vlastním kódem. Jakmile uživatel napíše kód, může ho stisknutím tlačítka "Validace"zaslat na server pro zpracování. Server projde uživatelův kód a buď vygeneruje příslušnou třídící síť, nebo vrátí uživateli chybovou hlášku popisující, kde se nachází chyba.

Kód je vepisován uživatelem do text boxu, který se nachází v kontrolním panelu vpravo. Aby uživatel mohl odsazovat text pomocí tabu, musí být každý nový znak odchycen a zkontrolován. Použití klávesy *Tab* by normálně způsobilo opuštění textboxu. Takhle je na požadované místo vložen znak odsazení a nedojde k opuštění text boxu.



Obrázek 7: Aktivitní diagram pohybu objektu třídy NumberCart

Pro zpracování kódu byl vytvořen parser, který zpracovává textový řetězec. Kód není na straně serveru nijak spouštěn a pouze se vytvoří objekty přímo určené k následujícímu vytvoření třídící sítě. Nemůže tak dojít k žádnému spuštění nežádoucího kódu na straně serveru.

6.3.1 Jazyk

Jazyk sloužící k tvorbě sítí je inspirován jazyky jako např. *C#* nebo *JavaScript*. Byl definován tak, aby obsahoval to nejdůležitější, co by uživatel mohl potřebovat k vytvoření třídící sítě, a zároveň nebyl přehlcen příliš velkým množstvím možností.

Vstupním místem programu je funkce sort s parametry *int[] array* (představující pole určené k setřídění) a *int length* (hodnota popisující velikost pole). Proměnné mohou být typu číselné (*int*) nebo logické hodnoty (*bool*).

Nejdůležitějším prvkem jazyka je funkce *swap*, která slouží k vytvoření komparátoru na třídící síti. Tato funkce přijímá tři parametry, pole a dva indexy označující dráty, mezi kterými

bude vytvořen komparátor.

Jazyk nadále umožňuje použití *for* cyklů. Každý další blok obklopen složenými závorkami i v případě, že se tam nachází pouze jeden řádek kódu. To kromě zjednodušení vytvořené gramatiky popisující tento jazyk také zlepšuje přehlednost pseudokódu pro uživatele.

6.3.2 Bezkontextová gramatika

Gramatika použita pro zpracování uživatelova pseudokódu není LL(1), protože se rozhoduje podle více než jednoho znaku. Gramatika je implementována jako třída *Grammar*, která spolupracuje s třídou *Scanner*.

$$
\begin{aligned}
S &\rightarrow functionID(int[]id, intid)A \\
A &\rightarrow \{B\} \\
B &\rightarrow CB|DB|EB;|FB;|\epsilon \\
C &\rightarrow for(F;G;E)A|for(E;G;E)A \\
D &\rightarrow swap(H,H); \\
E &\rightarrow J = I|J++|J-- \\
F &\rightarrow intid|int[]id|boolid|bool[]id|intid = I|int[]id = I|boolid = G|bool[]id = G \\
G &\rightarrow !GK|(G)K|I < IK|I > IK|I <= IK|I >= IK|I == HK|I! = GK \\
H &\rightarrow id|id[I] \\
I &\rightarrow L|L + I|L - I|L * I|L/I|L\%J \\
J &\rightarrow id|id[number]|id[id] \\
K &\rightarrow \&\&|'||'|\epsilon \\
L &\rightarrow id|number
\end{aligned}
$$

Objektu třídy *Grammar* je na začátku předán pseudokód ve tvaru textu a jeho úkolem je ho zpracovat. Jeho jednotlivé funkce (kromě funkce sloužící pro samotné spuštění zpracovávání) odpovídají daným výrazům bezkontextové gramatiky.

Nejmenším prvkem pseudokódu je tzv. Token. Tokeny představují jednotlivé typy slov a znaků. Mezi Tokeny patří např. slova *function*, *int* nebo speciální znaky jako levá kulatá závorka.

Pro průchod textem je použit objekt třídy *Scanner*, který postupně prochází pseudokód. *Scanner* vždy pracuje najednou pouze s jedním Tokenem. Pokud se jedná o Token, který má navíc nějakou důležitou hodnotu (např. se jedná o číslo nebo název proměnné), *Scanner* si kromě tokenu uloží i danou hodnotu. Pokud se jedná o znaky, které se mezi Tokeny nenachází, *Scanner* nahlásí chybu a průchod pseudokódem je označen za neúspěšný.

Objekt třídy *Grammar* funguje na úrovni celých výrazů. Přes třídu *Scanner* získává jednotlivé Tokeny, pomocí kterých prochází gramatikou. Jelikož se jedná o implementaci LL(1)

gramatiky, v každém kroku *Grammar* používá pouze jeden Token a vždy je další krok jasný. Pokud žádná cesta nepovoluje daný Token, průchod gramatiky je zrušen a pseudokód je označen jako nesprávný.

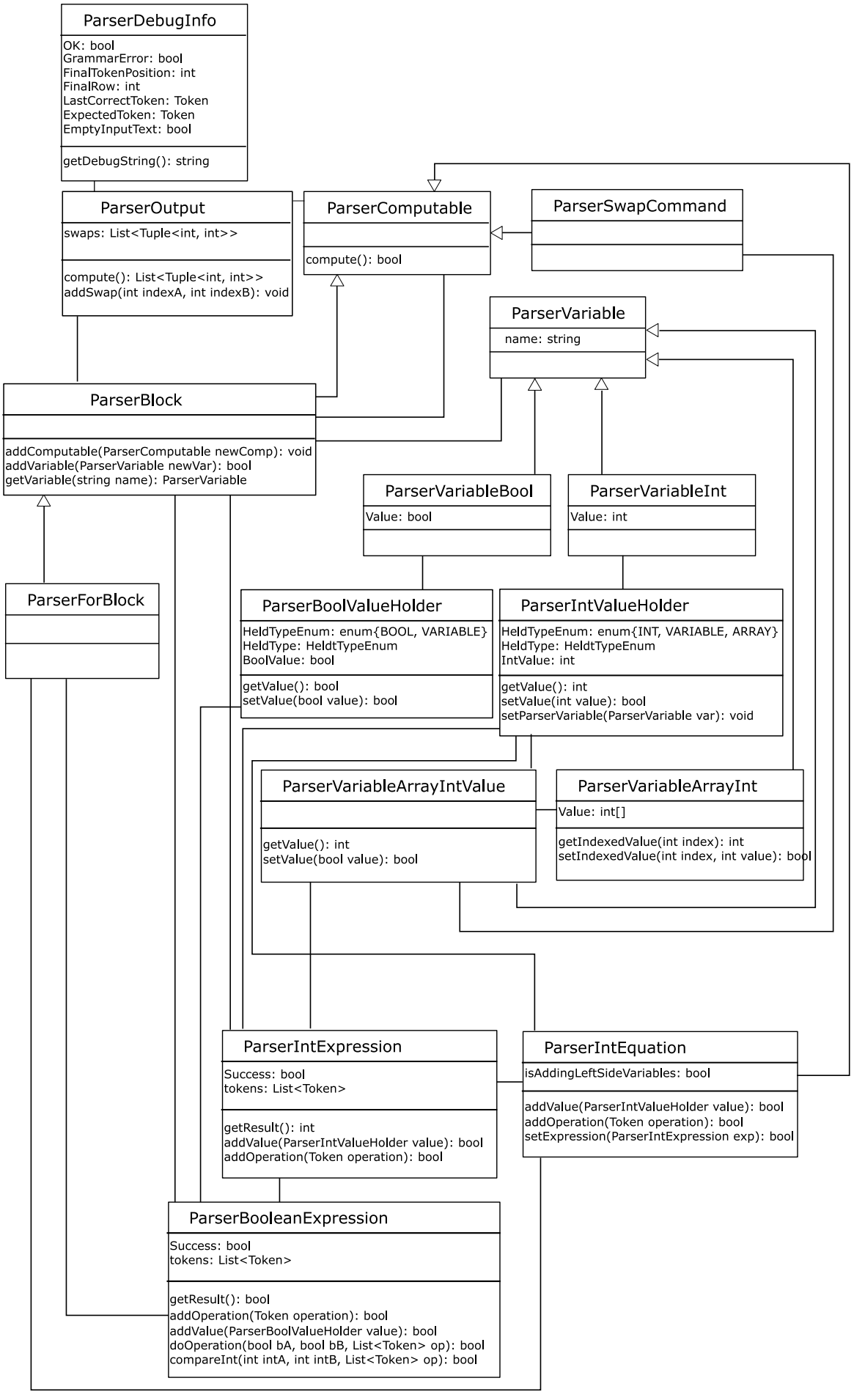
6.3.3 Vytvoření sítě

V průběhu validace pseudokódu objekt třídy *Grammar* navíc vytváří strukturu objektů popisující daný pseudokód. V průběhu validace pseudokódu jsou využívány pro validaci věcí jako např. názvy proměnných.

Tato struktura je zabalená do objektu třídy *ParserOutput*. Tato třída slouží jako zastřešení celé struktury objektů, které představují jednotlivé prvky kódu. Tyto objekty se dají rozdělit na dvě velké skupiny: objekty vykonávající kód a objekty držící data. Do první skupiny patří objekty které simulují chování pseudokódu. Jedná se např. o objekty simulující *for* cyklus nebo vypočítavající matematický výraz. Druhá skupina obsahuje objekty, které mají za úkol držet informace o proměnných, jako např. jejich hodnotu, typ a název.

Veškeré informace popisující úspěch nebo neúspěch parsování uživatelova pseudokódu jsou uloženy v objektu třídy *ParserDebugInfo*. V případě neúspěchu je tomuto objektu nastaven typ chyby s případnými dalšími informacemi (např. řádek, na kterém došlo k chybě). Následně je uživateli zobrazen text o úspěchu či neúspěchu parsování, který je tímto objektem vygenerován.

Jakmile je gramatika zvalidována, je zpátky vrácena odpovídající struktura zastřešená objektem třídy *ParserOutput*. Na této struktuře je následně spuštěna simulace vykonávání kódu. V průběhu vykonávání je kromě další validace (např. kontrola nekonečných cyklů) také vytvářeno pole komparátorů. Pokud bylo vykonávání úspěšné, vytvořené pole komparátorů se vloží do grafu.


ParserDebugInfo
OK: bool
GrammarError: bool
FinalTokenPosition: int
FinalRow: int
LastCorrectToken: Token
ExpectedToken: Token
EmptyInputText: bool
getDebugString(): string
ParserOutput
swaps: List<Tuple<int, int>>
compute(): List<Tuple<int, int>>
addSwap(int indexA, int indexB): void
ParserComputable
compute(): bool
ParserSwapCommand
ParserVariable
name: string
ParserBlock
addComputable(ParserComputable newComp): void
addVariable(ParserVariable newVar): bool
getVariable(string name): ParserVariable
ParserVariableBool
Value: bool
ParserVariableInt
Value: int
ParserForBlock
ParserBoolValueHolder
HeldTypeEnum: enum{BOOL, VARIABLE}
HeldType: HeldtTypeEnum
BoolValue: bool
getValue(): bool
setValue(bool value): bool
ParserIntValueHolder
HeldTypeEnum: enum{INT, VARIABLE, ARRAY}
HeldType: HeldtTypeEnum
IntValue: int
getValue(): int
setValue(int value): bool
setParserVariable(ParserVariable var): void
ParserVariableArrayIntValue
getValue(): int
setValue(bool value): bool
ParserVariableArrayInt
Value: int[]
getIndexedValue(int index): int
setIndexedValue(int index, int value): bool
ParserIntExpression
Success: bool
tokens: List<Token>
getResult(): int
addValue(ParserIntValueHolder value): bool
addOperation(Token operation): bool
ParserIntEquation
isAddingLeftSideVariables: bool
addValue(ParserIntValueHolder value): bool
addOperation(Token operation): bool
setExpression(ParserIntExpression exp): bool
ParserBooleanExpression
Success: bool
tokens: List<Token>
getResult(): bool
addOperation(Token operation): bool
addValue(ParserBoolValueHolder value): bool
doOperation(bool bA, bool bB, List<Token> op): bool
compareInt(int intA, int intB, List<Token> op): bool

6.4 Evaluátor třídící sítě

Na straně serveru se nachází třída *SortEvaluator*, která slouží ke zpracování a ohodnocení třídící sítě. Síť je předána objektu třídy *SortEvaluator* ve formátu JSON. Pro převod do pole komparátorů je použit NuGet balíček Newtonsoft.JSON. Jeho úkolem je danou třídící síť zpracovat a zjistit, zda daná třídící síť je validní (dokáže setřídit libovolně setříděný vstup dané velikosti). V případě, že třídící síť je validní, tak jsou vygenerovány statistiky o síti, které kromě informování uživatele slouží také k výpočtu skóre, které danou síť popisuje.

6.4.1 Testování sítě

Získané pole komparátorů je otestováno tříděním všech variant jedniček a nul využitím 0-1 principu (viz Podkapitolu 2.6). Každé unikátně setřízené pole jedniček a nul je třízeno testovaným polem komparátorů. Komparátory se prochází v daném pořadí (pole komparátorů je zadáno již správně seřazené) a pro každý z nich jsou porovnány dané hodnoty v poli jedniček a nul.

Jakmile jsou použity všechny komparátory, zkontroluje se setřízenost tříděného pole. Pokud je pole nesprávně setřízené, dojde k ukončení testování s tím, že zadaná třídící síť není validní. Pokud je pole setřízené, vygeneruje se nové pole jedniček a nul. Toto pole se vytvoří inkrementálně v závislosti na předchozím poli. Pole se zleva doprava prochází, dokud se nenalezne pozice s nulou. Hodnota této pozice je změněna na jedničku a všechny hodnoty vlevo od této jsou změněny na nulu. Vytváření nového pole je velmi podobné binární inkrementace, akorát je jednička přidána vlevo a ne vpravo.

V případě, že jedna varianta jedniček a nul nebude úspěšně setřízená, bude daná varianta uložena. Podle ní se následně vygeneruje pole čísel s hodnotami od 1 do n, kde n je velikost pole. Toto pole je setřízené tak, aby odpovídalo struktuře nesetříděného pole jedniček a nul. Toto vygenerované pole je předáno grafu na klientské straně. Uživatel následně uvidí třdění tohoto pole, které bude neúspěšné, a bude upozorněn na selhání.

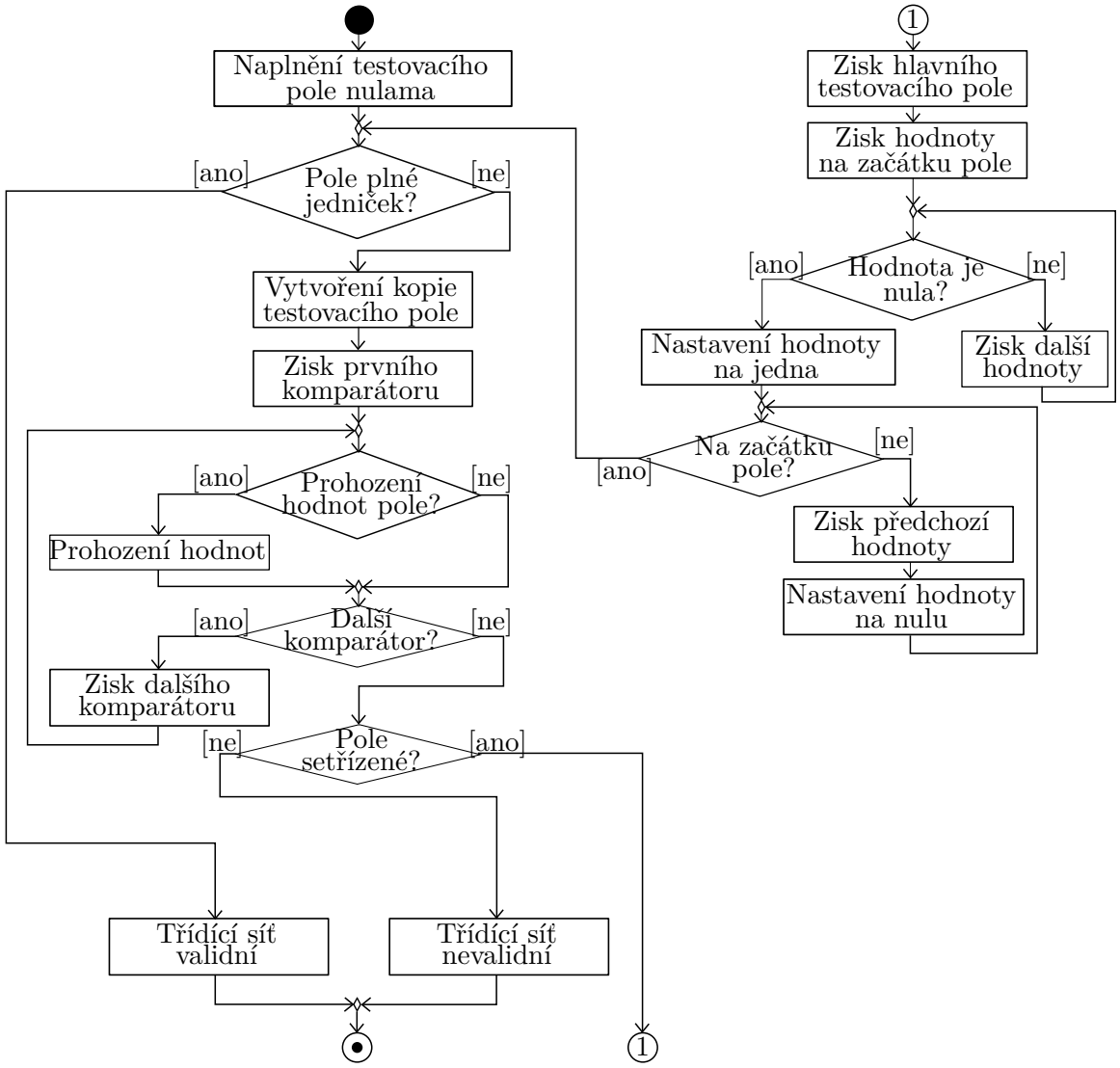
V případě úspěšné validace sítě bude třízené pole náhodně vygenerované na klientské straně. Uživatel v takovém případě uvidí úspěšné třdění.

Testování sítě je výpočetně nejnáročnější část celé aplikace. Jak bylo zmíněno v Podkapitole 2.6, složitost 0-1 principu, který je zde implementován, je $n!$, kde n je velikost pole, přestože se jedná o nejlepší způsob testování třídících sítí. Proto se nedoporučuje nastavovat příliš velkou velikost vstupu třídící sítě.

6.4.2 Statistiky sítě

V průběhu validace třídící sítě objekt třídy *SortEvaluator* vygeneruje statistiky o předané třídící síti. Mezi tyto statistiky patří:

- Počet použitých komparátorů
- Počet kroků algoritmu



Obrázek 8: Aktivitní diagram testování validace tídící sítě objektem třídy *SortEvaluator*

Počet použitých komparátorů odpovídá velkosti vnitřně uloženého pole komparátorů, které bylo objektu třídy *SortEvaluator* předáno na začátku. Počet kroků algoritmu se získává z atributu „Pozice “posledního komparátoru ve stejném poli. Zde se využívá toho, že komparátory v poli jsou vždy vzestupně seřazeny podle atributu Pozice.

Tyto statistiky jsou zahozeny v případě nevalidní třídící sítě. Pokud je však třídící síť validní, tyto statistiky jsou zobrazeny uživateli, a je z nich vygenerované finální skóre.

6.4.3 Bodové ohodnocení sítě

Objekt třídy *SortEvaluator* po úspěšné validaci třídící sítě vygeneruje bodové ohodnocení. Pro jeho výpočet jsou použity statistiky, které byly vygenerovány při validaci sítě. Každá statistika

má svoje bodové ohodnocení, ze kterých je složeno finální skóre. Finální skóre se vypočítává následujícím vzorcem:

$$x = s * 50 + c * 25 \tag{1}$$

kde s je celkoý počet kroků algoritmu a c odpovídá celkovému počtu použitých komparátorů.

Vygenerované skóre je nižší, čím lepší je výsledná třídící síť. Pokud bychom chtěli častěji používané skóre, tedy čím více bodů, tím vyšší skóre, bylo by nutné skóre, tak jak je nyní vypočítáváné odečítat od předem definované konstanty. Zde se však vynořují dva problémy. První je existence velmi důležité „magické“ konstanty, které by mohlo způsobit zmatení uživatele. Druhým důvodem je skutečnost, že v případě velmi špatné třídící sítě by mohlo dojít k zisku negativního skóre. K tomu nyní nemůže dojít, jelikož aktuální hodnoty, se kterými se počítá, nemohou dosáhnout záporných hodnot.

6.5 Komunikace klienta se serverem

O řízení komunikace mezi klientem a serverem se stará samotný ASP.NET, není potřeba implementovat vlastní způsob komunikace. Komunikace začíná vždy na klientově straně vyvoláním eventu některého z *WebControls*. Tento event je pak zpracován na serveru. Každá *WebControl* má vlastní event, který určí, co se bude dít.

Problém nastal s uložením pole komparátorů. Komparátory jsou potřeba přeposílat mezi klientem a serverem ze dvou hlavních důvodů. Za prvé jsou při parsování uživatelova pseudokódu vytvářeny na serveru a klientská strana je potřebuje zobrazit. Za druhé server potřebuje zpracovat hotovou třídící síť a ohodnotit ji. Třetím důvodem je, že v průběhu hry je stránka několikrát obnovována (např. při validaci pseudokódu) a klientská strana si z jednotlivých instancí nepamatuje pole komparátorů, takže je potřeba uložit je bokem.

K vyřešení těchto problémů je využita *WebControls* komponenta typu „hidden“. Tato komponenta se nijak nevykresluje (je neviditelná pro uživatele) a dá se do ní uložit libovolný textový řetězec. Jako *WebControls* komponenta si navíc pamatuje svůj stav mezi obnovováními stránky. Všechny tyto vlastnosti z této komponenty činí perfektní řešení pro problém předávání pole komparátorů.

Pole komparátorů je neustále ukládáno v hidden komponentě. Pro uložení pole komparátorů v *hidden* a jednoduchému předávání mezi klientem a serverem je pole uloženo ve formátu JSON. Na straně klienta jsou používány vestavěné příkazy pro převod do a z JSON formátu. Na straně serveru je používán NuGet Newtonsoft.JSON.

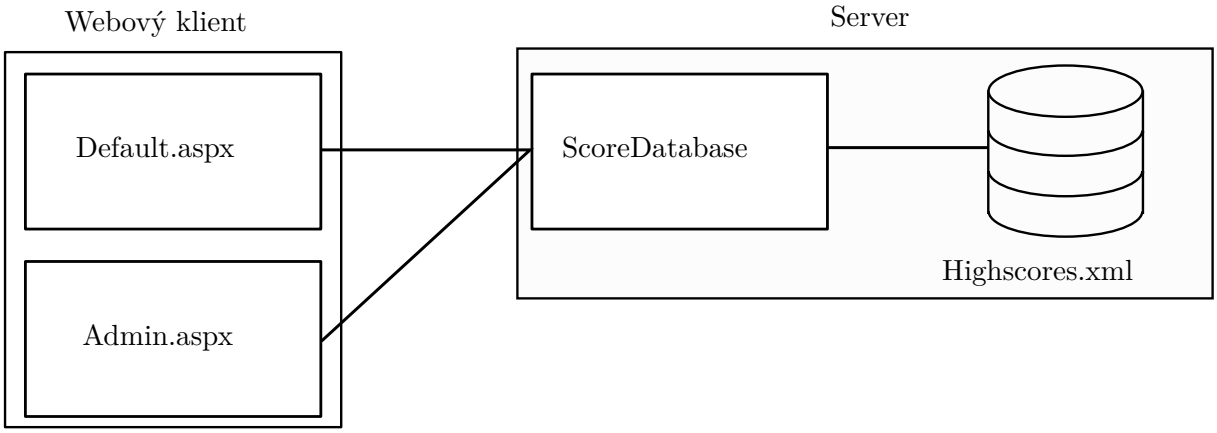
Na straně klienta je pole komparátorů v *hidden* komponentě ukládáno při každé změně pole. Takhle je vždy připravená aktuální záloha pro případ, že by došlo k obnovení stránky. Klient se po každém novém načtení stránky podívá do hidden komponenty a nahraje si pole komparátorů v případě, že se nějaké pole v *hidden* komponentě nachází. V opačném případě je pole prázdné.

6.6 Databáze skóre

Při úspěšné validaci třídící sítě je síť ohodnocena a uživatel je požádán o jméno, které bude přidáno do tabulky. Databáze je implementována jako jeden XML soubor obsahující jméno uživatele a jeho přezdívku.

Přístup do databáze je řešen přes objekt třídy *ScoreDatabase*. Pro jednodušší přeposílání skóre mezi objekty je navíc definována třída *Score*, která obsahuje jméno uživatele a jeho počet bodů. Třída *ScoreDatabase* zpracovává potřebné úkoly na databázi:

- Zisk veškerého skóre
- Přidání nového skóre
- Odstranění konkrétního skóre
- Celkové vyčištění tabulky skóre



Obrázek 9: Diagram komunikace klienta s databází skóre

Pro práci s XML souborem používá *ScoreDatabase* objekty třídy *XmlDocument*. Ten oproti kombinaci tříd *XmlReader* a *XmlWriter* umožňuje jak čtení souboru, tak zapisování. Této vlastnosti je využito hlavně při přidávání nového skóre do tabulky. *XmlDocument* je použit i při zisku vekerého skóre, jelikož i když není potřeba zapisovat do XML souboru, je jednodušší používat jednu metodu práce s XML souborem všude, a tou je použití *XmlDocument*.

Skóre v XML souboru jsou uloženy seřazené vzestupně podle počtu bodů. Při vkládání nového skóre je skóre vloženo na správnou pozici XML souboru, aby hodnoty zůstaly seřazené. To zjednodušuje vypisování skóre, jelikož není nutné při každém zbrazení hodnoty seřazovat.

6.7 Vícejazyčnost aplikace

Aplikace podporuje dva plně podporované jazyky, ve kterých může vypisovat informace: angličtina a čeština. Výběr aktivního jazyka je nastaven automaticky podle kultury uživatelova

zařízení. Jako výchozí kultura je nastavena angličtina, takže pokud bude kultura zařízení jiná než čeština nebo nebude existovat český překlad určitého textu, bude použitý anglický ekvivalent.

Pro každý jazyk je použitý tzv. *Resource*, který obsahuje data ve formátu klíč/hodnota. V kódu samotném je pouze použitý klíč prezentující požadovaný text. Server pak na základě aktuální kultury zvolí správný *Resource*, ze kterého bude použitý vypsaný text. V případě, že daný *Resource* neobsahuje požadovaný klíč, je automaticky použitý primární *Resource*.

Rozdělení překladů na jednotlivé *Resource* nabízí množství výhod. Všechny hodnoty jsou uloženy na jednom místě, což umožňuje jednodušší údržbu. Jelikož je výběr jazyku automatický podle kultury zařízení, tak je jednoduché přidat nový jazyk (stačí se dodržet správné pojmenovávací konvence).

7 Závěr

V rámci této práce byla vytvořena webová aplikace prezentující problém třídících algoritmů. Tato webová aplikace bude reprezentovat skupinu Teoretické Informatiky na dnech otevřených dveří. Důležitou součástí práce tedy bylo vymyslet způsob, jak třídící algoritmy prezentovat zábavnou formou.

Základem pro vytvoření této aplikace bylo seznámit se třídícími sítěmi. Třídící sítě byly zvoleny pro prezentaci problému třídících algoritmů protože přehledně dokážou vizualizovat třízení čísel a jejich tvorba je pro případného začátečníka jednoduchá. Kromě funkčnosti třídících sítí jsem se také seznámil s jejich teorií, od složitosti sítí přes paralelizaci a jednoduchou tvorbu až k jejich praktickému použití.

Webová aplikace umožňuje napsat vlastní kód, pomocí kterého je následně vygenerována třídící síť. K tomu jsem se musel seznámit s teorií bezkontextových gramatik, konkrétně LL(1) gramatikou. Následně jsem vytvořil vlastní bezkontextovou gramatiku popisující jazyk sloužící ke generování třídících sítí. Tato gramatika je implementována a propojena se strukturou tříd popisující daný algoritmus. Tato struktura je vytvořená tak, aby imitovala spuštění kódu tak, jak by měl proběhnout, s třídící sítí jako výstupem.

Mezi případná zlepšení aplikace patří umožnění vytvoření rekurzivního kódu uživatelem a definování vlastních funkcí. Dále kód neumožňuje používat v podmínkách hodnoty třízeného pole, jelikož v době parsování nejsou hodnoty známé. Dále pro lepší výkon aplikace na straně klienta by bylo žádoucí přeimplementovat vykreslování canvasů, aby se vykreslovaly na požádání oproti pravidelnému intervalu. Nakonec by bylo potřeba uložit heslo administrátora zašifrovaně a ne jako čistý text.

Literatura

[1] Time Complexities of all Sorting Algorithms. Geeks for Geeks [online]. Noida [cit. 2018-03-19]. Dostupné z: https://www.geeksforgeeks.org/time-complexities-of-all-sorting-algorithms/

[2] CARMEN, Thomas H., Charles E. LEISERSON a Ronald L. RIVEST. CHAPTER 28: SORTING NETWORKS. Intro to Algorithms [online]. [cit. 2018-04-04]. Dostupné z: http://staff.ustc.edu.cn/~csli/graduate/algorithms/book6/chap28.htm

[3] KNUTH, Donald Ervin. The art of computer programming. 3rd ed. Reading, Mass.: Addison-Wesley, c1997. ISBN 0-201-89685-0.

[4] HOYTE, Doug. Introduction to Sorting Networks [online]. [cit. 2018-04-04]. Dostupné z: https://hoytech.github.io/sorting-networks/

[5] Donald E. Knuth. The art of computer programming. 3rd ed. Reading, Mass.: Addison-Wesley, c1997, s. 223. ISBN 0-201-89685-0.

[6] Sorting Networks [online]. Hochschule Flensburg, 1997, 05.03.1997 [cit. 2018-03-26]. Dostupné z: http://www.iti.fh-flensburg.de/lang/algorithmen/sortieren/networks/sortieren.htm

[7] Context-Free Grammar Introduction. Tutorials Point [online]. Tutorials Point, c2018 [cit. 2018-04-24]. Dostupné z: https://www.tutorialspoint.com/automata_theory/context_free_grammar_introduction.htm

[8] MAZA, Marc Moreno. LL(1) Grammars. Compiler Theory: Syntax Analysis [online]. 2004, 2 December 2004 [cit. 2018-04-24]. Dostupné z: http://www.csd.uwo.ca/~moreno/CS447/Lectures/Syntax.html/node14.html

[9] FIRST and FOLLOW sets: a necessary preliminary to constructing the LL(1) parsing table [online]. October 8, 2007 [cit. 2018-04-11]. Dostupné z: https://web.eecs.umich.edu/~weimerw/2008-415/reading/FirstFollowLL.pdf

[10] ASP.NET Overview. Microsoft Developer Network [online]. Redmond: Microsoft, ©2018 [cit. 2018-02-28]. Dostupné z: https://msdn.microsoft.com/en-us/library/4w3ex9c2.aspx

[11] ANDERSON, Rick, Luke LATHAM, Samir PATEL, Andy PASIC a Tom DYKSTRA. ASP.NET Overview. Microsoft Docs [online]. Redmont: Microsoft, ©2018, 03/12/2010 [cit. 2018-02-28]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/overview

[12] ANDERSON, Rick, Andy PASIC a Tom DYKSTRA. What is Web Forms. Microsoft Docs [online]. Redmont: Microsoft, ©2018, 02/21/2014 [cit. 2018-02-28]. Dostupné z: https://docs.microsoft.com/en-us/aspnet/web-forms/what-is-web-forms

[13] WAGNER, Bill, Luke LATHAM, Petr ONDERKA a Maira WENZEL. A Tour of C#. Microsoft Docs [online]. Redmont: Microsoft, ©2018, 08/10/2016 [cit. 2018-02-28]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/

[14] HTML Basics. MDN Web Docs [online]. Mozilla, ©2005-2018,[cit. 2018-02-28]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics

[15] HTML 5.2. World Wide Web Consortium [online]. World Wide Web Consortium, 2017 [cit. 2018-03-05]. Dostupné z: https://www.w3.org/TR/2017/REC-html52-20171214/introduction.html#html-vs-xhtml

[16] HTML5. MDN Web Docs [online]. Mozilla, ©2005-2018 [cit. 2018-02-28]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5

[17] HTML5 Introduction. W3Schools [online]. Refsnes Data, ©1999-2018 [cit. 2018-03-02]. Dostupné z: https://www.w3schools.com/html/html5_intro.asp

[18] How CSS works. MDN Web Docs [online]. Mozilla, ©2005-2018 [cit. 2018-02-28]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/CSS/Introduction_to_CSS/How_CSS_works

[19] CSS3. MDN Web Docs [online]. Mozilla, ©2005-2018 [cit. 2018-02-28]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS3

[20] JavaScript Guide: Introduction. MDN Web Docs [online]. Mozilla, ©2005-2018 [cit. 2018-02-28]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction

Příloha

Přiložené CD obsahuje:

- Text bakalářské práce ve formátu pdf
- Zdrojové soubory webové aplikace
- Textový soubor s přístupovým heslem do administrace aplikace